

Stored Procedures with Transactions (PostgreSQL)

Transactions

A transaction is a sequence of database operations that must all succeed or all fail. Transactions protect data consistency, especially when updating multiple tables. In PostgreSQL stored procedures (not functions) you can use Commit and Rollback to save or reverse any changes. A new transaction is started automatically after a transaction is ended using these commands, so there is no separate Begin Transaction command.

Transaction Syntax:

Statement	Description
-----------	-------------

Commit	Saves all changes made during the transaction
--------	---

Rollback	Reverses all changes made during the transaction
----------	--

The default behavior for the PL/pgSQL body is atomicity and if the body cannot complete, it is rolled back before entering the exceptions block. When an error is caught by an Exception clause, the local variables remain as they were when the error occurred, but all changes to persistent database state within the block are rolled back.

Create a procedure PR_AddStudentPayment that inserts a payment record in the Payment table and reduces the Balance Owing value in the Student table. Use a transaction to ensure both succeed or both fail. Input parameters are Payment ID, Payment Date, Payment Amount, PaymentTypeID, and Student ID.

Create Procedure PR_AddStudentPayment

```
(p_PaymentID integer, p_PaymentDate date, p_Amount decimal(6,2),
p_PaymentTypeID integer, p_StudentID integer)
Language plpgsql
As $body$
Begin
    Begin
        Insert Into Payment
            (PaymentID, PaymentDate, Amount, PaymentTypeID, StudentID)
        Values
            (p_PaymentID, p_PaymentDate, p_Amount, p_PaymentTypeID,
p_StudentID);
        Raise Notice 'Payment % inserted successfully for student %', p_PaymentID,
            p_StudentID;
    Exception When Others Then
        Rollback ;
        Raise Exception 'Insert Payment % failed with Error: %', p_PaymentID,
SQLERRM;
    End;
    Begin
        Update          Student
        Set              BalanceOwing = BalanceOwing - p_Amount
        Where            StudentID = p_StudentID;
        Raise Notice 'Payment % applied successfully for student %', p_PaymentID,
            p_StudentID;
    Exception When Others Then
        Rollback ;
        Raise Exception 'Update Student % failed with Error %', p_StudentID,
SQLERRM;
    End;
    Commit;
End;
$body$;
```

To Test and Run:

Select PaymentID, PaymentDate, Amount, PaymentTypeID, StudentID from Payment
where PaymentID = 34;

Select StudentID, BalanceOwing from Student where StudentID = 200122100;

Call PR_AddStudentPayment (34, Current_Date, 450.00, 1, 200122100);

Select PaymentID, PaymentDate, Amount, PaymentTypeID, StudentID from Payment
where PaymentID = 34;

Select StudentID, BalanceOwing from Student where StudentID = 200122100;

Optional: Drop the procedure and re-create with default values of null.

Drop Procedure PR_AddStudentPayment (p_PaymentID integer, p_PaymentDate
date, p_Amount decimal(6,2), p_PaymentTypeID integer, p_StudentID integer);

Create Procedure PR_AddStudentPayment

(p_PaymentID integer = Null, p_PaymentDate date = Null, p_Amount decimal(6,2) = Null, p_PaymentTypeID integer = Null, p_StudentID integer = Null)

Language plpgsql

As \$body\$

Begin

If p_PaymentID Is Null Or p_PaymentDate Is Null Or p_Amount Is Null Or
p_PaymentTypeID Is Null Or p_StudentID Is Null Then

Raise Notice 'You must enter a Payment ID, Payment Date, Payment Amount,
PaymentTypeID, and Student ID';

Else

Begin

Insert Into Payment

(PaymentID, PaymentDate, Amount, PaymentTypeID,
StudentID)

Values

(p_PaymentID, p_PaymentDate, p_Amount, p_PaymentTypeID,
p_StudentID);

Raise Notice 'Payment % inserted successfully for student %', p_PaymentID,
p_StudentID;

Exception When Others Then

Rollback ;

Raise Exception 'Insert Payment % failed with Error: %', p_PaymentID,
SQLERRM;

End;

Begin

Update Student

Set BalanceOwing = BalanceOwing - p_Amount

Where StudentID = p_StudentID;

Raise Notice 'Payment % applied successfully for student %',
p_PaymentID,

p_StudentID;

Exception When Others Then

Rollback ;

Raise Exception 'Update Student % failed with Error %', p_StudentID,
SQLERRM;

End;

Commit;

End If;

End;

`$body$;`

To Test and Run:

-- Run with no input arguments

Call PR_AddStudentPayment ();

-- Run with good data

Select PaymentID, PaymentDate, Amount, PaymentTypeID, StudentID from Payment
where PaymentID = 36;

Select StudentID, BalanceOwing from Student where StudentID = 200122100;

Call PR_AddStudentPayment (36, Current_Date, 450.00, 1, 200122100);

Select PaymentID, PaymentDate, Amount, PaymentTypeID, StudentID from Payment
where PaymentID = 36;

Select StudentID, BalanceOwing from Student where StudentID = 200122100;
values